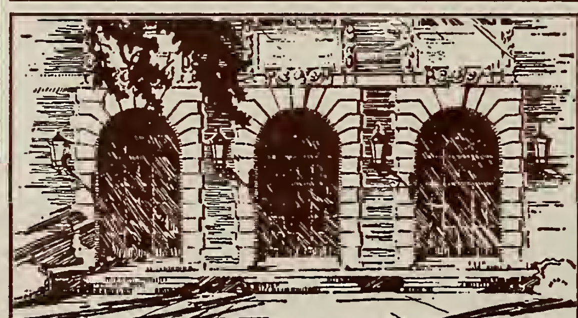


[redacted]

Math

Report No. 438

A BRANCH-AND-BOUND ALGORITHM FOR OPTIMAL NOR NETWORKS*
(The Algorithm Description)

by

Tomoyasu Nakagawa
Hung-Chi Lai

April 1971

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

Report No. 438

A BRANCH-AND-BOUND ALGORITHM FOR OPTIMAL NOR NETWORKS*
(The Algorithm Description)

by

Tomoyasu Nakagawa
Hung-Chi Lai

Department of Computer Science
University of Illinois
Urbana, Illinois 61801

* This work was supported in part by the National Science Foundation under Grant No. NSF GJ-503

CONTENTS

Introduction

The branch-and-bound algorithm is applied by E.S. Davidson to the synthesis problem of optimal combinational networks for arbitrary switching functions using NAND gates, by introducing the desirability order and other speed improvement gimmicks [1]. (In our expository paper [2], we summarized his algorithm, explaining the details of these gimmicks.) We implemented his algorithm for NOR gates by using simpler heuristics than Davidson's and also by incorporating a new gimmick called 'redundancy check'. This report presents the description of our version of the branch-and-bound method.

1. Definitions and the Basic Form of the Algorithm

We want to solve the synthesis problem of optimal combinational networks with NOR gates for m Boolean functions of n variables.

The criterion of optimality is the minimization of the cost function C defined by $C = A \times R + B \times I$, where R is the number of gates, I is the total number of inputs to gates (i.e., the sum of connections of external variables and interconnections among gates), and A, B are non-negative coefficients (i.e., weights). Different combinations of the weights A and B imply different optimization problems: $A > 0$ and $B = 0$ implies the minimization of the number of gates, $A \gg B > 0$ implies the minimization of the number of gate inputs after first minimizing the number of gates, and so on. In this report, however, we will be concerned with the case $A \gg B > 0$ unless we mention otherwise.

In the algorithm, we will represent given m output functions of n external variables in terms of a truth table. Let x_ℓ, $\ell = 1, \ldots, n$, be n external variables, and let f_h, $h = 1, \ldots, m$, be the given m output functions. The x_ℓ, $\ell = 1, \ldots, n$, and f_h, $h = 1, \ldots, m$, are expressed in the following way:

$$\left.\begin{array}{lll} & x_\ell = (x_\ell^0, \ldots, x_\ell^{2^n-1}), & \ell = 1, \ldots, n \\ \text{and} & f_h = (f_h^0, \ldots, f_h^{2^n-1}), & h = 1, \ldots, m \end{array}\right\} \quad (1.1).$$

For example, in the case of $m = 1$, if the output function f_1 is $\bar{x}_1 x_3 \vee x_1 \bar{x}_2 \bar{x}_3$, then

$$\left.\begin{array}{ll} & x_1 = (\,0\;0\;0\;0\;1\;1\;1\;1\,) \\ & x_2 = (\,0\;0\;1\;1\;0\;0\;1\;1\,) \\ & x_3 = (\,0\;1\;0\;1\;0\;1\;0\;1\,) \\ \text{and} & \\ & f_1 = (\,0\;1\;0\;1\;1\;0\;0\;0\,) \end{array}\right\} \quad (1.2).$$

We will exclude from our discussion the case where some of the output functions among the m functions are identical to other functions or external variables. Therefore, we always need at least m gates to realize m output functions f_h. Let us assign one NOR gate labeled h to each f_h, $h = 1, \ldots, m$. We call this network (i.e., m isolated gates whose outputs are assigned f_h, $h = 1, \ldots, m$) the <u>initial solution</u>. Fig. 1.1 is the initial solution to the problem of an output function f_1, $f_1 = \bar{x}_1 x_3 \vee x_1 \bar{x}_2 \bar{x}_3$, for a special case of $m = 1$.

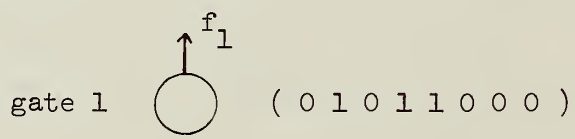


Fig. 1.1 The initial solution to $f_1 = \bar{x}_1 x_3 \vee x_1 \bar{x}_2 \bar{x}_3$. The output of the NOR gate is assigned f_1, but the gate has no inputs yet.

The algorithm starts with the initial solution, and expands the initial solution by connecting external variables, by introducing new gates, or by making interconnections among gates, so that the resulting loop-free networks realize the m given functions simultaneously.

In accordance with (1.1), the output of every gate to be introduced into a network is represented in the form of 2^n-tuple as $(P^0, \ldots, P^{2^n-1})$, where each P^j for $j = 0, \ldots, 2^n-1$, may assume the values 0 or 1. It should be noted, however, that the algorithm will not assign a definite value 0 or 1 at once to all components, P^j's, of a gate. Accordingly, we use the symbol $*$ to denote the value of P^j is unassigned. Let us find out a necessary condition that the output $(P^0, \ldots, P^{2^n-1})$ of any gate in a NOR network satisfies. Take two gates, i and k. Let $(P_i^0, \ldots, P_i^{2^n-1})$ and $(P_k^0, \ldots, P_k^{2^n-1})$ denote the outputs of gate i and gate k, respectively. If gate i is connected

to gate k, then the components P_i^j and P_k^j must satisfy the following condition, no matter whether or not gate i and gate k have other inputs, because the gates perform NOR operation:

$$\left.\begin{aligned} & P_k^j = 0 \quad \text{for all } j \text{ such that } P_i^j = 1 \\ \text{and} \quad & P_i^j = 0 \quad \text{for all } j \text{ such that } P_k^j = 1 \end{aligned}\right\} \qquad (1.3).$$

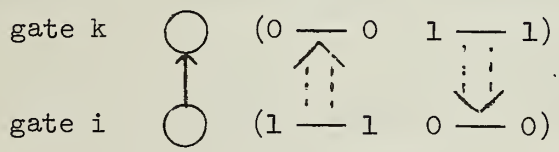


Fig. 1.2. If there are 1-components in a NOR gate, then corresponding components in its immediately preceding/succeeding gates must be 0.

A similar condition must also hold between the output of gate k and an external variable x_ℓ, when x_ℓ is connected to gate k, regardless of other inputs to gate k:

$$\left.\begin{aligned} & P_k^j = 0 \quad \text{for all } j \text{ such that } x_\ell^j = 1, \\ \text{and} \quad & x_\ell^j = 0 \quad \text{for all } j \text{ such that } P_k^j = 1 \end{aligned}\right\} \qquad (1.4).$$

If the assignment of binary values to the components P^j's of the output $(P^0, \ldots, P^{2^n-1})$ of a gate satisfies the above condition with respect to all of immediately preceding/succeeding gates and all connected external variables, then we call this assignment of $(P^0, \ldots, P^{2^n-1})$ a <u>feasible assignment</u>.

(Notice that some of components P^j's could be *.)

Using the concept of feasible assignment, let us define an "intermediate solution".

Definition (Intermediate solution).

A network of R gates, $R \geq m$, with external variables x_ℓ is called an intermediate solution if the network satisfies the following set of conditions:

(i) The entire network has no loops. R gates are numbered 1 through R, as gate 1, ..., gate R.

(ii)-a The first m gates (i.e., gate i, for i = 1, ..., m) are assigned m output functions. In other words, the output of gate i is $(f_i^0, \ldots, f^{2^n-1})$, for i = 1, ..., m. These gates may or may not be connected to other gates yet.

(ii)-b The outputs of the remaining gates (i.e., gate i, i m + 1, ..., R), if any, are completely or incompletely specified. Each gate i for i = m + 1, ..., R, is connected to at least one of other gates in the network.

(iii) The assignment of the output of each gate is feasible.

Notice that the initial solution defined previously is a special case of an intermediate solution.

The network corresponding to an intermediate solution may or may not realize the given set of functions f_h, h = 1, ..., m. An intermediate solution whose network realizes the given set of functions is said to be a feasible solution. A feasible solution whose cost is the least among all feasible solutions is an optimal solution.

In order to construct feasible solutions, we introduce the concept of "cover".

Definition (Covered component and uncovered component)

A component $P_k^{j_o} = 0$ of gate k is said to be covered, if gate k has at least one input (i.e., the output of gate i or external variable x_ℓ) whose j_o-th component ($P_i^{j_o}$ or $x_\ell^{j_o}$) is 1. $P_k^{j_o} = 0$ of gate k is said to be uncovered, if $P_k^{j_o}$ is not yet covered.

Fig. 1.3 is an example of intermediate solution, where some components are already covered. (The covered components are shown with underlines.)

Clearly an intermediate solution is a feasible solution if all output components $P_k^j = 0$ in all gates k are covered.

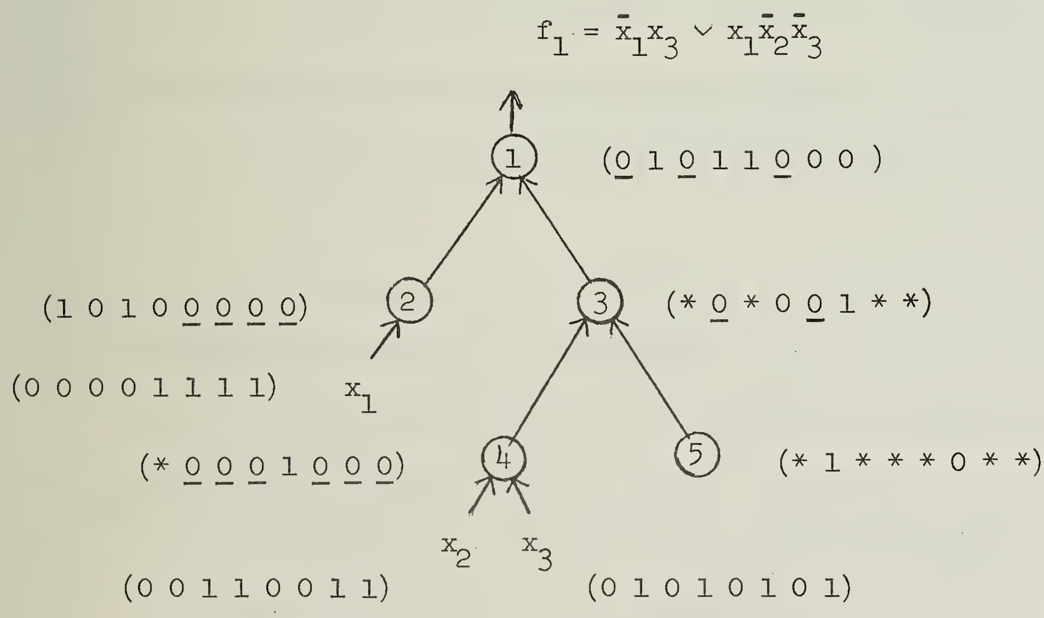


Fig.1.3 Example of an intermediate solution for $f = \bar{x}_1 x_3 \vee x_1 \bar{x}_2 \bar{x}_3$, where some components are covered and others are not. (The covered components are underlined.)

Let us introduce the concept of possible covers for an uncovered component. As seen from the definition below, possible covers are the only available external variables and/or gates with which we can cover the uncovered component under consideration. Suppose $P_k^{j_o}$ in gate k is an uncovered component in a given intermediate solution.

<u>Definition (Possible covers of an uncovered component $P_k^{j_o} = 0$ of gate k)</u>

(I) An external variable x_ℓ which is not yet connected to gate k is a possible cover of $P_k^{j_o} = 0$ if x_ℓ satisfies the following condition:

$$\begin{cases} x_\ell^{j_o} = 1, & \text{and} \\ x_\ell^{j} = 0 & \text{for all } j \text{ such that } P_k^j = 1. \end{cases}$$

(II) A gate i which is already connected to gate k is a possible cover of $P_k^{j_o} = 0$ if gate i satisfies the following condition:

$$P_i^{j_o} = *.$$

(III) A gate i which is not yet connected to gate k is a possible cover of $P_k^{j_o} = 0$ if gate i satisfies the following condition:

$$\begin{cases} \text{a connection of gate } i \text{ to gate } k \text{ will not form any loops,} \\ P_i^{j_o} = 1 \text{ or } *, \text{ and} \\ P_i^{j} = 0 \text{ or } * \quad \text{for all } j \text{ such that } P_k^j = 1. \end{cases}$$

(IV) A gate which is not yet incorporated in the intermediate solution is a possible cover. This gate is called <u>a new gate</u>, and satisfies the following condition:

The output components are all *.

The gate number of this gate is assigned R + 1, when the highest gate number in the current intermediate solution is R.

With each of the possible covers defined above, we can cover $P_k^{j_o}$ according to the following procedure, called an implementation of a possible cover.

Procedure (Implementation of a possible cover)

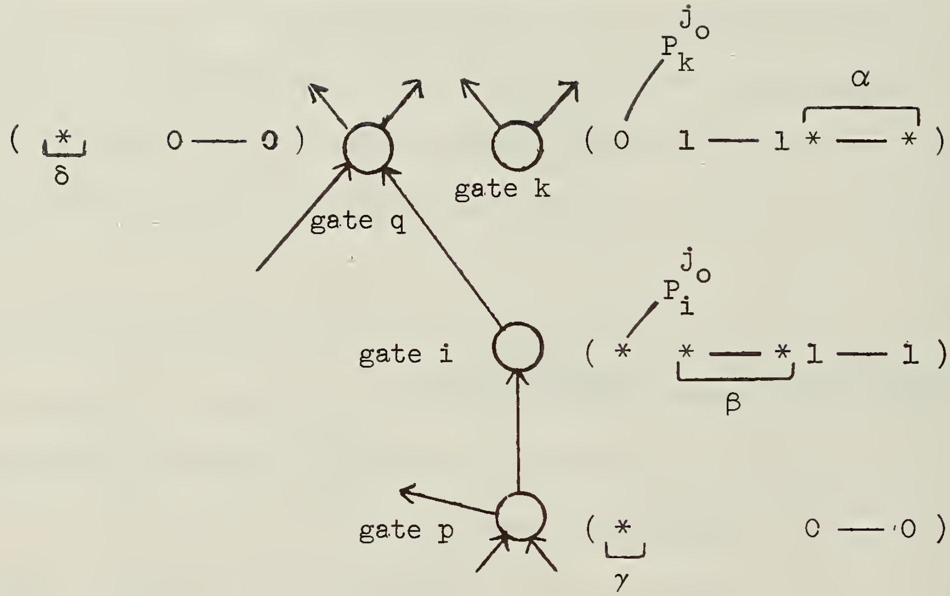
Step 1: If the possible cover is not yet connected to gate k, then connect it to gate k; otherwise do nothing.

Step 2: If the j_o-th component of the possible cover is not yet assigned, then assign the value 1 to it; otherwise do nothing.

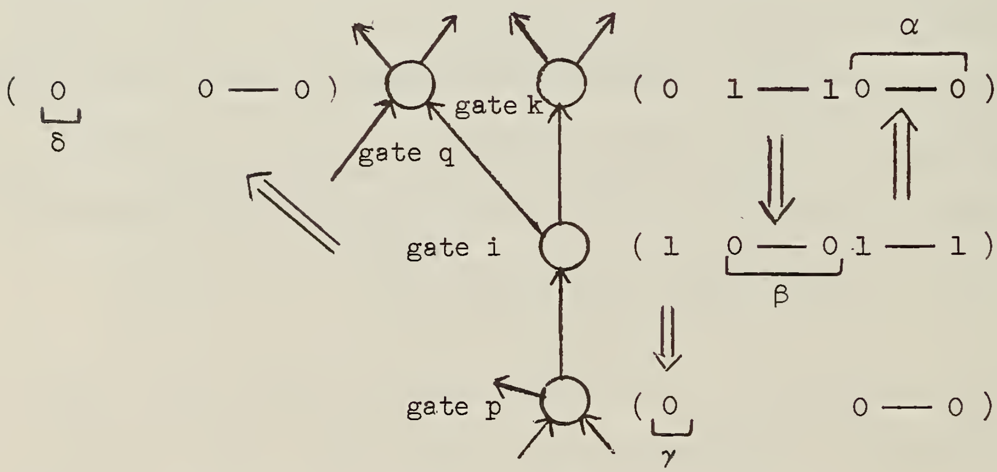
Step 3: Assign the value 0 to unassigned components so that the assignment of the output of every gate in the network be feasible.

The Fig. 1.4 illustrates the above procedure for the case where a possible cover is gate i of definition (III) above.

Suppose we are going to cover $P_k^{j_o}$ in gate k with gate i in the figure (a). By step 1, we connect gate i to gate k. By step 2, we set $P_i^{j_o} = 1$. And by step 3, we assign the value 0 to the components indicated by α, β, γ, and δ. The resulting network is the figure (b).



(a) before covering of $P_k^{j_o} = 0$ with gate i.



(b) after a covering of $P_k^{j_o} = 0$ with gate i

Fig. 1.4 Illustration of the procedure of implementing a possible cover

By applying the above procedure repeatedly to uncover components in each of the intermediate solutions, we eventually obtain intermediate solutions in which all output components $P_k^j = 0$ in all gate k are covered. In order to enumerate all such intermediate solutions systematically, we introduce the following set of rules.

Definition (SCUC & IPPC)

The selection criterion of uncovered components (SCUC) is the criterion under which an uncovered component $P_k^j = 0$ is selected from the given intermediate solution. The implementation priority of possible covers (IPPC) is the priority under which the order of implementation among the possible covers for the selected uncovered component is determined.

Using the concepts defined in this section, we present the basic form of Davidson's branch-and-bound algorithm based on which he made several versions of programs. In the algorithm below, we use a parameter $\bar{C}$ which we call the cost ceiling, or the incumbent cost; the $\bar{C}$ is used to preclude all intermediate solutions whose cost exceeds the cost of the current best feasible solution. Initially $\bar{C}$ is set to a sufficiently large number.

The basic form of the algorithm (Davidson)

Step 0 (start): $k = 1$.

Let S_1 denote the initial solution.

Set $\bar{C}$ to a sufficiently large number.

Step 1: Calculate the cost C_k of the current intermediate solution S_k.

Compare C_k with $\overline{C}$. If C_k is greater than $\overline{C}$[†], then go to step 7-1; otherwise go to step 2.

Step 2: Search for an uncovered component in S_k. If there is none, then go to step 8; otherwise go to step 3.

Step 3: Select one uncovered component from S_k, according to the selection criterion of uncovered component (SCUC). Let $\hat{P}$ denote it.

Step 4: Make a list of all possible covers of $\hat{P}$.

Step 5: Store $\hat{P}$ and the list of possible covers in a working space, and label k to this portion of working space. Select one possible cover from the list, according to the implementation priority of possible covers (IPPC).

Step 6: Increment k by 1.

Implement the possible cover selected at step 5, generating the augmented intermediate solution, S_k. Go to step 1.

Step 7 (backtrack)

Step 7-1: Decrease k by 1.

If k becomes 0, then go to step 9; otherwise go to step 7-2.

Step 7-2: Retrieve $\hat{P}$ and the list of possible covers of the label k from the working space.

Search for unimplemented possible covers in this list.

If there are no unimplemented possible covers, then go to step 7-1; if there are some, then go to step 7-3.

Step 7-3: Reconstruct S_k.

Select one of the unimplemented possible covers from the list, according to the IPPC.

---

† If we replace this condition '$C_k > \overline{C}$' by '$C_k \geq \overline{C}$', then the algorithm obtains only one optimal network, instead of all optimal networks.

Step 7-4: Increment k by 1.

Implement the possible cover taken at step 7-3, generating the augmented intermediate solution, S_k. Go to step 1.

Step 8 (solution): Print S_k.

Replace the value of $\overline{C}$ with the cost C_k of S_k.

Go to step 7-1.

Step 9: Stop.

In the next two sections we describe two kinds of gimmicks to obtain our improved version of the algorithm: our modification of the set of SCUC and IPPC, and a redundancy check of feasible solutions which prunes non-optimal solutions. We modify Davidson's versions of the algorithm with these gimmicks. The entire description of the algorithm is presented in section 4.

2. Heuristics for the Set of SCUC and IPPC

The set of SCUC and IPPC is the most important part of the algorithm, because this determines the order of intermediate solutions which the algorithm enumerates. The investigation on this part of the algorithm was the subject with which Davidson was mostly concerned; he experimented eight versions of his program with different combinations of SCUC's and IPPP's. According to the experiment, he chose the best version among the eight. However, the heuristics incorporated into the SCUC and IPPC of this version (and also most of other versions) is fairly complicated.

We conceived a set of SCUC and IPPC which is simpler than the simplest version of Davidson's program, and which yet works as comparably well as his most elaborate version.

Before presenting our version of the SCUC and IPPC, we define some concepts* necessary for describing the SCUC and IPPC.

Type of covered component

A component $P_k^{j_o} = 0$ which is already covered is assigned the type COV. (COV stands for a component which is already COVered.)

Types of possible covers of an uncovered component $P_k^{j_o} = 0$ of gate k.

Possible covers† are classified into the following seven types.

G^*: A gate i which is already connected to gate k, and has $P_i^{j_o} = *$.

(G^* stands for a Gate having $P_i^{j_o} = *$)

---

* See [2].
The concept defined here is essentially the same as Davidson's. However, we use different mnemonic names for the types in the concept.

† For the definition of possible cover, see Section 1.

VC^*: An external variable x_ℓ which is not yet connected to gate k, and which has $x_\ell^{j_o} = 1$, and $x_\ell^j = 0$ for all j such that $P_k^j = 1$. (VC^* stands for a Variable whose Connection being * (unassigned)).

GC^*O: A gate i which is not yet connected to gate k, and which has $P_i^{j_o} = 1$, and $P_i^j = 0$ for all j such that $P_k^j = 1$. (GC^*O stands for a Gate whose Connection being *, and P_k^j = 0 for all j such that $P_k^j = 1$.)

GC^*O^*: A gate i which is not yet connected to gate k, which has $P_i^{j_o} = 1$, $P_i^j = 0$ or * for all j such that $P_k^j = 1$, and where there is at least one $P_i^j = *$ among those P_i^j's. (GC^*O^* stands for a Gate whose Connection is *, and which has $P_i^{j_o} = 1$, and P_i^j = 0 or * for all j such that $P_k^j = 1$.)

G^*C^*O: A gate i which is not yet connected to gate k, and which has $P_i^{j_o} = *$, and $P_i^j = 0$ for all j such that $P_k^j = 1$. (G^*C^*O stands for a Gate whose Connection is *, and which has $P_k^{j_o}$ = *, and P_i^j = 0 for all j such that $P_k^j = 1$.)

$G^*C^*O^*$: A gate i which is not yet connected to gate k, which has $P_i^{j_o} = *$, $P_i^j = 0$ or * for all j such that $P_k^j = 1$, and where there is at least one $P_i^j = *$ among those P_i^j's. ($G^*C^*O^*$ stands for a Gate whose Connection being *, $P_k^{j_o}$ = *, and P_i^j = 0 or * for all j such that $P_k^j = 1$.)

NWG: A new gate. (NWG stands for a NeW Gate.)

The desirability order of types is defined by

$$COV - G^* - VC^* - GC^*O - GC^*O^* - G^*C^*O - G^*C^*O^* - NWG$$

where COV is the most desirable, and NGW is the least desirable.

The type of an uncovered component is defined by the most desirable type among its possible covers.

The <u>type of gate k</u> is defined by the least desirable type among all components $P_k^j = 0$.

Based on these concepts, our version of the SCUC and the IPPC is as follows.

If the current intermediate solution has type NWG components, then we employ Davidson's special scheme, called 'Remove NF vectors'*, which generates from the original intermediate solution having type NWG components another intermediate solution in which no type NWG components exists. This scheme is to choose certain type NWG components to be covered in order to maximize the number of new gates which must be introduced into the network. This results in the preclusion of some equivalent networks with permuted gate labels. For details, refer to [2]. Therefore, we assume in the following that the current intermediate solution does not have uncovered components of type NWG.

<u>The SCUC</u>

If the current intermediate solution has uncovered components of types VC^* or less desirable, then select an uncovered component among those types such that it has the fewest possible covers†. If there are two or more uncovered components which has the same fewest number of possible covers, then select one whose type is the least desirable. If the current intermediate solution has only uncovered components of type G^*, then select an uncovered component (of type G^*) which has the fewest possible covers.

---

* This is explained in [2], as 'Special treatment of type NWG components'.

† Davidson chooses the least desirable type but did not choose the type of the fewest possible covers.

The IPPC

Implement the possible covers of the selected uncovered component according to the following order

$$G^* - VC^* - \{GC^*O,\ GC^*O^*,\ G^*C^*O,\ G^*C^*O^*\}^\dagger - NWG.$$

The possible cover(s) of type G^* are assigned the highest priority and the possible cover of type NWG is assigned the lowest priority.

Davidson did not use the types of possible covers as the primary criterion in his IPPC.

The motivation of establishing this set of the SCUC and IPPC is as follows.

An uncovered component which has less number of possible covers is "hard to cover" in a sense, since if we postpone covering of this uncovered component until later, this uncovered component will more likely lose any possible covers except a new gate. Accordingly we might miss good networks. Therefore it seems a good rule to cover first an uncovered component which has the fewest possible covers in a given intermediate solution. However, if we take simply "the fewest possible covers" as the main criterion of selecting uncovered components without distinguishing type G^* components from uncovered components of other types, then type G^* components could be selected too frequently at an early stage of the branching steps. Concentration on covering type G^* components may more likely result in the following situation: one gate which is introduced as a new gate to cover a certain component of another gate realizes the negation of the latter gate. With repetition of this process, we may have indefinitely cascaded inverters without completing a feasible network. In order to avoid such a hazard, we select

---

† [illegible]

one with the fewest possible covers among those whose types are VC^* or less desirable. When the given intermediate solution has only uncovered components of type G^*, we select one with the fewest possible covers among them.

Once an uncovered component is selected, we cover this uncovered component with each of its possible cover, generating the corresponding augmented intermediate solution. We have to order these possible covers such that we generate a 'good' augmented intermediate solution first, i.e., one which is as close to a feasible solution as possible. The desirability order seems one of good rules for this purpose.

The difference of our heuristics from Davidson's may be illustrated in the following way. The entire searching process of the branch-and-bound algorithm is represented by a tree (called a search tree) in which the root node corresponds to the initial solution of the problem, and non-terminal nodes and terminal nodes correspond to intermediate solutions and feasible solutions, respectively. Davidson's SCUC is based on minimizing the length of paths from the root node to terminal nodes. Therefore, the search tree corresponding to his SCUC is shallow in depth, but it may become broad in width. On the other hand, our SCUC is based on minimizing the number of branches originating from each node. Therefore the entire search tree corresponding to our heuristics is generally thin in width.

3. Pruning Non-Optimal Solutions by Redundancy Check

Because of the nature of the branch-and-bound method, the first feasible solution obtained by the algorithm may not be optimal. After finding the first feasible solution, the algorithm searches for better feasible solutions with the use of backtracking. Given the cost ceiling $\bar{C}$, the algorithm discards all intermediate solutions whose cost exceed the cost ceiling, which results in pruning of many non-optimal solutions implicitly. The lower the cost ceiling, the more the algorithm prunes non-optimal solutions. The redundancy check stated in [4] is a new gimmick which applies transformations to the current feasible solution in order to get a better solution. Thus, the cost ceiling can be lowered and some feasible solutions would be pruned. The following is the summary of this gimmick.

(I) Elimination of gates

I-(i) We search for external variables and/or outputs of gates whose disjunction turns to be identical to the output of gate i under consideration.

If we find such external variables and/or output of gates, then we replace each of the output connections of gate i by the new connections of those external variables and/or gates (Fig. 3.1).

I-(ii) We search for external variables and/or outputs of gates whose new connections in some portion of the entire network make gate i having only one output connection redundant.

Since finding such external variables and/or outputs of gates in general case is extremely difficult, we investigate several special cases. The following is one among those we investigated.

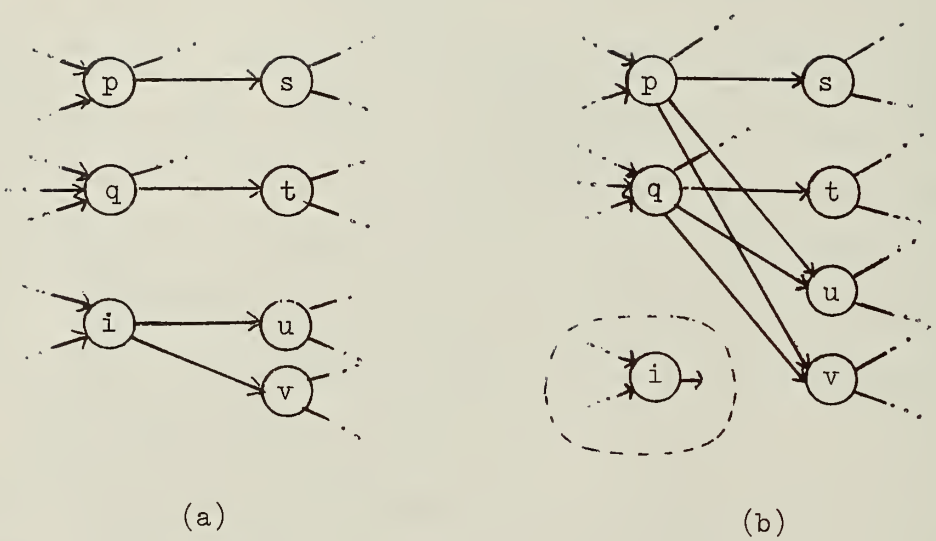


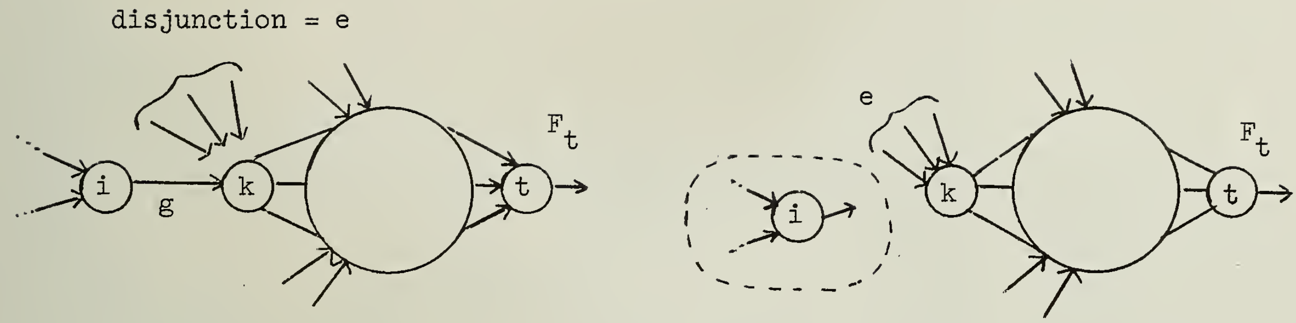
Fig. 3.1 Example of Transformation I-a: If the output of gate i is identical to the disjunction of the outputs of gates p and q in (a), then we have a network having one less gate as shown in (b).

Assume that the only output g of gate i is connected to gate k. Let gate t denote the output gate of the entire network, and let F_t be its output. Define $h^*_{kt} = F_t \oplus F'_t$, and $e^*_{kt} = \overline{F_t \oplus F''_t}$, where F_t is the output of gate t when g is disconnected from gate k, and F''_t is the output of gate t when the constant input 1 is connected to gate i. If we find some external variables and/or gates whose disjunction e satisfies $h^*_{kt} \subseteq e \subseteq e^*_{kt}$, then we eleiminate gate i and its input connections and output connection, after connecting these external variables and/or gates to gate k (Fig. 3.2).

Other transformations similar to the above one are explained in [4].

(II) Elimination of Connections

II-(i) We search for connections (of external variables or gate outputs) whose disconnections do not change the output of the entire network. If we find such connections, then we eliminate them.



(a) (b)

Fig. 3.2 If e satisfies $h^*_{kt} \subseteq e \subseteq e^*_{kt}$ in (a), then we have a better network as shown in (b).

II-(ii) We search for an external variable or the output of a gate whose new connection to another gate makes some existing connections redundant. The transformation is based on the following property of a NOR network configuration as shown in Fig. 3.3 (a).

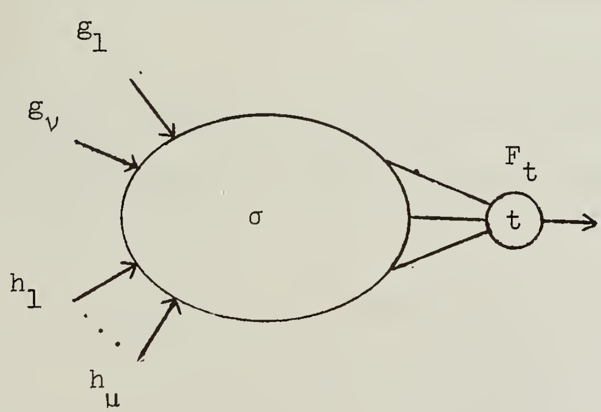


Fig. 3.3(a) A NOR network configuration which consists of a subnetwork σ and gate t, where all output connections of σ go to only gate k, but not to other gates.

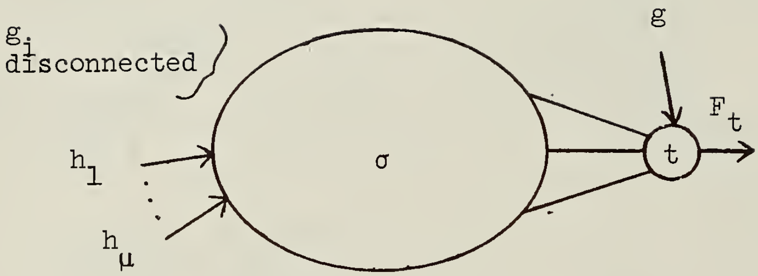


Fig. 3.3(b) A network configuration equivalent to Fig. 3.3(a).

Property

If an external variable g (or the output g of a gate outside of the σ) satisfies $\bar{F}_t \geq g$, along with $g \geq g_i$, for $i = 1, \ldots, \nu$, where g_i are some of inputs to the σ, then Fig. 3.3(b) is an equivalent network configuration with respect to the output of gate t. Fig. 3.3(b) has $(\nu-1)$ less inputs than Fig. 3.3(a).

By combining the above transformations, we have the computational procedure for checking redundances as shown in Fig. 3.4.

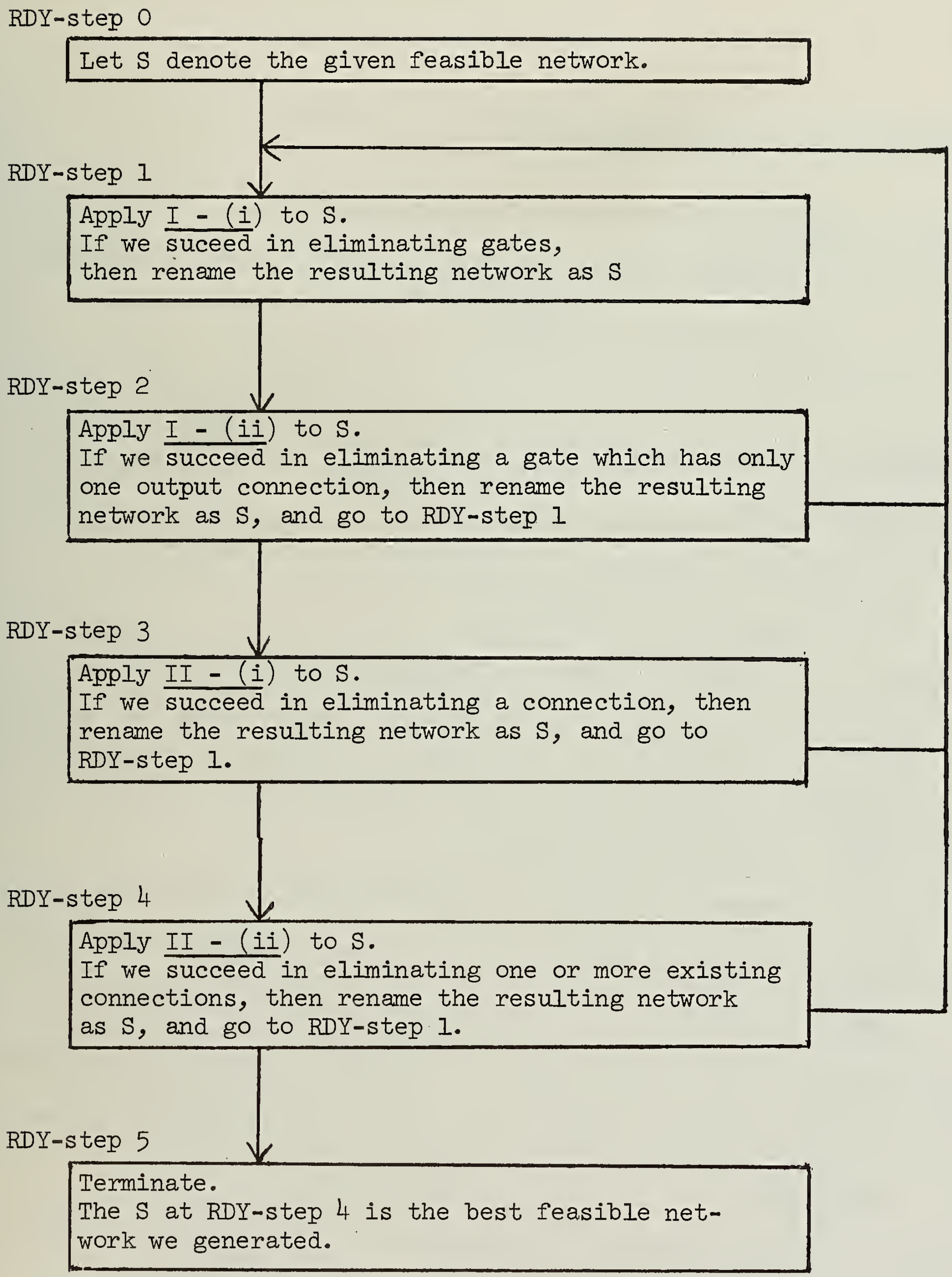


Fig. 3.4 The flow chart of the redundancy check procedure.

4. Description of the Entire Algorithm

Incorporating the heuristics of the SCUC and IPPC and the redundancy check into the basic form of the branch-and-bound algorithm, we complete our version of the algorithm as follows.

Step 0 (start): Set $K = 1$, and $BTK = 0$. (BTK counts the number of backtracks.)

Set the cost ceiling $\bar{C}$ to a sufficiently large number.

Set up the initial solution S_1.

Step 1 (loop): Calculate the cost C_k of S_k; the cost is defined by $A \times R + B \times I$, where R is the number of gates, I is the number of inputs to gates, and A and B are the given non-negative weights.

Step 2: Search for uncovered components.

If the current intermediate solution S_k has no uncovered components, then go to step 8 (solution); if S_k has no type NWG components, then go to step 3; if S_k has type NWG components, apply to S_k the scheme of 'Special treatment of type NWG components'; the resulting intermediate solution has no type NWG components. Rename this intermediate solution as S_k. Go to step 3.

Step 3: Select one uncovered component from S_k, with the use of the selection criterion of uncovered components (SCUC).

Let $\hat{P}$ denote the selected uncovered component .

Step 4: List all possible covers of $\hat{P}$.

Calculate the cost $\tilde{C}$ for each possible cover; the cost $\tilde{C}$ is defined by $C_k + \Delta C_k + c$, where C_k is the cost of S_k, $\Delta C_k = B \times \{$the number of gates which are other than the gate containing the $\hat{P}$, and whose

types are different from COV and G^*}, and

$$c \begin{cases} = 0, \text{ if the type of the possible cover is } G^* \\ = B, \text{ if the type of the possible cover is one of } VC^*,\ GC^*O, \\ \qquad GC^*O^*,\ G^*C^*O, \text{ and } G^*C^*O^*, \\ = A + 2 \times B, \text{ if the type of the possible coves is NWG.} \end{cases}$$

Accept only the possible covers for which $\tilde{C} \leq \bar{C}$ holds, then sort those remaining according to the implementation priority of possible covers (IPPC), and go to step 5; in the case where there is no acceptable possible cover, go to step 7 (backtrack).

<u>Step 5</u>: Make a double-entry list (i.e., entries: a possible cover and the corresponding $\tilde{C}$ for each item) of the accepted possible covers such that the first item is the possible cover of the highest priority and its corresponding $\tilde{C}$, the second item is the possible cover of the second highest priority and its corresponding $\tilde{C}$ ..., and the last item is the possible cover of the lowest priority and its corresponding $\tilde{C}$. Store the $\hat{P}$ and the double-entry list into a working space. Label K to this portion of working space. We call this working space <u>the possible cover list (PC-list)</u>, and we refer to the $\hat{P}$ and the double-entry list as the K-th $\hat{P}$ and the K-th block of the PC-list, respectively. Select the first possible cover from the K-th block of the PC-list. Go to step 6.

<u>Step 6</u>: Increment K by 1.

Implement the first possible cover selected at step 5, generating the augmented intermediate solution, S_K.

Go to step 1.

Step 7 (backtrack): BTK = BTK + 1.

Step 7-1: Decrease K by 1.

If K becomes 0, then go to step 9; otherwise go to step 7-2.

Step 7-2: Retrieve the K-th $\hat{P}$ and the K-th block of the PC-list.

If there are no unimplemented possible covers at all, or the cost $\tilde{C}$ of every unimplemented possible cover exceeds the cost ceiling $\bar{C}$, then go to step 7-1; otherwise go to step 7-3.

Step 7-3: Reconstruct S_K.

Select from the K-th block of the PC-list the possible cover which was implemented last time. If the type of this possible cover is G^*, then set the j_o-th component of gate i to 0, i.e., $P_i^{j_o} = 0$, where j_o is the component position of the $\hat{P}$ and i is the gate number under consideration. (Notice the possible cover is a gate, since the type of the possible cover is G^*.) Go to step 7-4; if the type of this possible cover is one of VC^*, GC^*0, or GC^*0^*, then impose a condition that this possible cover (an external variable or a gate) be prohibited from connecting to the gate that contains the K-th $\hat{P}$, in any succeeding intermediate solutions. Go to step 7-4. If the type of the possible covers is different from G^*, VC^*, GC^*0, or GC^*0^*, then do nothing.

Go to step 7-4.

Step 7-4: Take from the K-th block of the PC-list the possible cover next to the possible cover taken at step 7-3. (This possible cover has the highest priority among unimplemented ones.)

Increment K by 1.

Implement this possible cover, generating the augmented intermediate solution, S_K. Go to step 1.

Step 8 (solution)

Step 8-1: Print S_K. Go to step 8-2.

Step 8-2 (redundancy check):

S_K is the current best network.

Apply to S_K the redundancy check procedure consisting of RDY-step 0 through RDY-step 5, as explained in section 1.

A new network at RDY-step 5, if any, is the current best network.

Let C^* denote the cost of this network.

If $C^* = \bar{C}$, then go to step 7-1; otherwise replace the value of $\bar{C}$ with C^*, and print the current best network.

Go to step 7-1.

Step 9 (stop)

Terminate the computation.

All the printed feasible solutions of the lowest cost are the optimal solutions.

REFERENCES

1. E.S. Davidson, "An algorithm for NAND decomposition of combinational switching functions," Ph.D. dissertation, Department of Electrical Engineering and Coordinated Science Laboratory, University of Illinois, 1968.

2. T. Nakagawa and S. Muroga,"Exposition of Davidson's thesis 'An algorithm for NAND decomposition of combinational switching systems,'" To be published, Department of Computer Science, University of Illinois.

3. T. Nakagawa and H.C. Lai, "Reference manual of FORTRAN program ILLOD-(NOR-B) for optimal NOR networks," To be published as a report, Department of Computer Science, University of Illinois.

4. T. Nakagawa, H.C. Lai and S. Muroga, "Pruning and branching methods for designing optimal networks by the branch-and-bound method," To be published as a report, Department of Computer Science, University of Illinois.

5. C.R. Baugh, T. Ibaraki, T.K. Liu and S. Muroga, "Optimum network design using NOR and NOR-AND gates by integer programming," Report No. 293, Department of Computer Science, University of Illinois, January 1969.

6. S. Muroga and T. Ibaraki, "Logical design of an optimum network by integer linear programming - Part I," Report No. 264, Department of Computer Science, University of Illinois, 1968.

7. S. Muroga and T. Ibaraki, "Logical design of an optimum network by integer linear programming - Part II," Report No. 289, Department of Computer Science, University of Illinois, December 1968.

8. S. Muroga,"Logical design of optimal digital networks by integer programming," in book Advances in Information Systems Science, Vol. 3 edited by J.T. Tou, Plenum Press, 1970, pp. 283-348.

9. T. Nakagawa and S. Muroga, "Comparison of the implicit enumeration method and the branch-and-bound method for logical design," To be published as a report, Department of Computer Science, University of Illinois.